A Tool Chain for the V&V of NASA Cryogenic Fuel Loading Health Management

Johann Schumann[1], Vanesa Gomez-Gonzalez[2], Nagabhushan Mahadevan[3], Michael Lowry[4], Peter Robinson[5], and Gabor Karsai[6]

[1] *SGT, Inc., NASA Ames Research Center, Moffett Field, CA 94035, USA*
johann.m.schumann@nasa.gov

[2] *USRA, Mountain View, CA 94043, USA*
vanesa.gomezgonzalez@nasa.gov

[3,6] *Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN 37212, USA*
nag,gabor@isis.vanderbilt.edu

[4,5] *NASA Ames Research Center, Moffett Field, CA 94035, USA*
{Michael.R.Lowry, Peter.I.Robinson}@nasa.gov

ABSTRACT

Complex machinery like spacecraft, aircraft, or chemical plants are equipped with fault detection and diagnosis systems. Due to their safety-critical nature, such diagnosis systems have to undergo rigorous Verification and Validation (V&V). In this paper, we present a tool suite to facilitate V&V of the deployed diagnostic system. The V&V relies on the paradigms of *cross validation* (to compare the diagnosis results of the deployed reasoner against those of other, more advanced reasoners), *automatic fault scenario generation* (to support extensive testing and coverage analysis), and *parametric model analysis* (to enrich test sets and for robustness and sensitivity analysis). We present the application of this tool architecture towards the V&V of the diagnosis system based on the TEAMS tool suite towards a subsystem in the NASA cryogenic fuel loading facility.

1. INTRODUCTION

Modern complex systems, like the NASA loading facility for cryogenic rocket fuel, are equipped with extensive fault detection and diagnosis systems to quickly detect off-nominal conditions and to diagnose faulty components. For the NASA Kennedy Cryo facility, the commercial TEAMS tool suite (`http://www.teamqsi.com`) is being used for modeling and diagnosis. Obviously, such a plant is highly safety-critical. Thus, fault detection and diagnosis must undergo rigorous V&V in order to ensure that the diagnostic system properly models the physical plant and any associated detectors, so as to minimize the number of false and missing alarms during operation.

In this paper, we present a tool architecture that has been designed to support V&V of TEAMS diagnostic models. Our modular set of tools allows the user to carry out a multitude of V&V use cases and is based upon three basic paradigms: *cross-validation*, *automatic fault scenario generation*, and *parametric analysis*. Our tools are augmented with report generators and a number of advanced statistical analysis and visualization capabilities.

Any diagnostic model is ultimately based on a simplified and abstracted model of the underlying physical plant. TEAMS/RT (Real Time) models are based upon multi-signal diagnosability analysis. Here, the outcome of individual tests ("pass", "fail", or "unknown") results in sets of components (or failure modes) known to be "good", "bad", "suspect", or "unknown", based upon an efficient algorithm using the model's diagnosability matrix (D-matrix). Because of its time-boundedness and efficiency, this kind of discrete diagnosis algorithm has become popular in the aerospace domain, although aspects of timing, fault propagation, fault probabilities, or physical model dynamics cannot be expressed. For real-time applications, the TEAMS/RT diagnosis engine is typically wrapped by custom code for data acquisition, discretization, and filters for noise and transient reduction. The V&V of this wrapper code is as critical as the V&V of the D-matrix. Timing

and fault propagation information, as well as data on component reliability is available in an extended TEAMS Designer model, which is used in an off-line mode for designing instrumentation. This additional information can also be obtained through information provided by subject-matter experts.

Our *cross-validation* tools use such additional information to facilitate deep model analysis. The provided TEAMS models are translated into a different, more expressive modeling paradigm (in our case Timed Failure Propagation Graphs and Bayesian networks) and are enriched with additional information. Then, failure scenarios are executed using reasoners for these paradigms, and results are compared and analyzed.

One of our reasoners is a Timed Failure Propagation Graph (TFPG) reasoner, which uses models that have been generated from the TEAMS models. These TFPG models capture the faults (failure modes), and their propagation effects to trigger one or more anomalies (tests). Additionally, the TFPG models can account for cascading effects of the failures, mode and timing constraints in the failure propagation, and additional information such as failure rate expressed in terms of Mean Time to Failure (MTTF). We also translate the TEAMS model's D-matrix into a Bayesian network (BN), which allows probabilistic diagnostic reasoning and the incorporation of priors on component reliability and failure likelihood.

Proper V&V requires the analysis of the health model to a certain degree of coverage and not just on a few selected and hand-crafted failure scenarios. While our tool set allows for manual specification of fault scenarios, it uses advanced algorithms to automatically generate single and multi-fault scenarios across the entire model or for a selected subset of faults. These scenarios are applied in the context of the mode-sequence commands prescribed in the operational test scripts for the plant. Our tool set uses the mode-enriched fault scenarios to generate the test/mode events from two independent streams—the discrete TFPG model (generated from TEAMS models) as well as a gold standard obtained from a Simulink plant simulation or from Cryo lab experimental data. Comparison of the data generated from the two independent streams allows for cross-validation of the discrete TFPG (TEAMS) model and the high-fidelity physics based Simulink model.

The V&V process is made more rigorous by perturbing a number of independent parameters including time of fault injection, fault magnitude, discretization, and thresholding parameters, among others. Parametric Model Analysis (PMA) provides a rich data set for a detailed analysis of the fault-effect coverage on the tests associated with the fault including analysis of the wrapper code.

This paper demonstrates the tool and its capability on a case study of a NASA cryogenic fuel loading facility.

This paper is structured as follows: after discussing related work, we will present our tool architecture (Section 3). In Section 4, will give a brief overview of the NASA cryogenic fuel loading facility and present a selected subsystem as our example. We then demonstrate sensitivity/robustness analysis, test/model coverage, and the analysis of cross-validation results. Section 5 concludes and discusses future work.

2. RELATED WORK

It is obvious that a fault detection and diagnosis system is a highly safety-critical piece of software. Thus, it needs to undergo rigorous V&V and certification. For example, DO-178C, Sec 2.4.3 (RTCA, 2011) requires that a monitoring device has to undergo V&V to the same level as the system it monitors. Due to its specific structure and the use of non-standard reasoning algorithms, however, traditional V&V techniques are not directly applicable, and only a few approaches toward V&V of fault detection and diagnosis systems have been reported. For example, Lindsey and Pecheur (2004) describe a model-checking approach for Livingston health models that can fully exercise the state space. Schwabacher, Feather, and Markosian (2008) discuss various approaches for the V&V of an advanced FDDR system for a NASA space system; Reed, Schumann, and Mengshoel (2011) describe an approach on systematic analysis (parametric analysis) of a Bayesian FDDR model for ADAPT.

As pointed out in (Schumann, Srivastava, & Mengshoel, 2010; Srivastava & Schumann, 2013), any diagnosis system must be analyzed and validated on both the *model level* and the *implementation level*. Most approaches in the literature aim at model validation; actual testing of the system implementation for code coverage (e.g., MC/DC (RTCA, 2011)) has not been reported yet and is difficult due to the usually table-driven algorithms in this domain. The approach described in this paper addresses both model-level and the implementation level validation—especially for key parameters of the wrapper code and the reasoner engine through cross-comparison with other reasoners.

3. TOOL ARCHITECTURE

Validation of the Systems Health Management (SHM) in safety critical systems through rigorous testing of the deployed diagnosis engines (reasoners) is extremely important for safety and mission success. This process should help to understand the quality and limitations of the current SHM setup and provide relevant guidance to further fine-tune and improve the performance of the health management system.

With this in mind, we have designed our tool suite (Figure 1) that uses the concepts of cross-validation to compare the results of the deployed baseline reasoner against other candidate reasoners that can employ richer models over a multitude of auto-generated test-cases (automatic fault scenario generation), taking into account the realistic variation of key

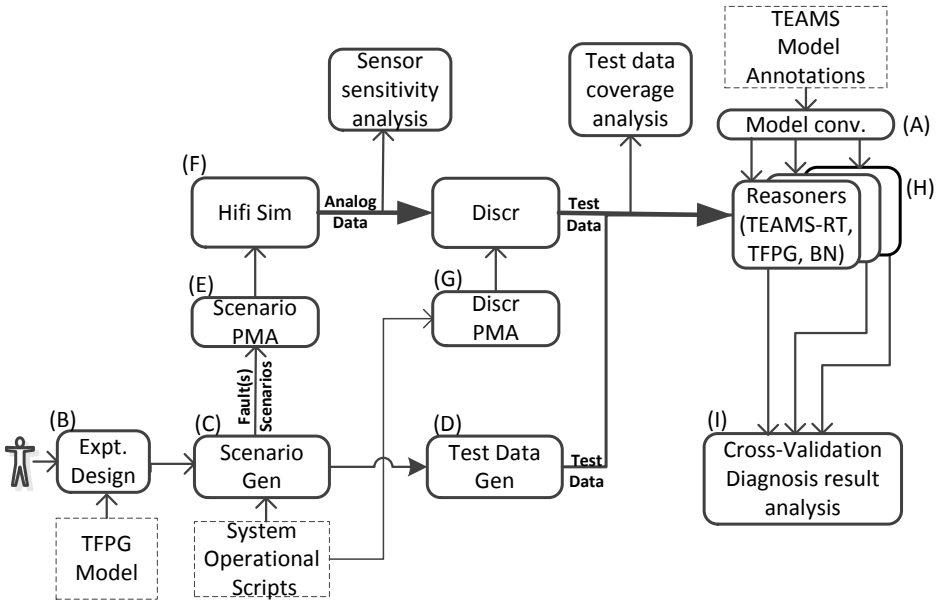

Figure 1. Tool architecture

parameters (parametric model analysis) related to the system (plant, signal preprocessing and discretization). Figure 1 captures the processes and flows across our V&V tool suite. The tool suite uses multiple model-based reasoners such as TEAMS/RT, TFPG, and Bayesian networks.

3.1. Overview

In an initial step (A), the given model, here developed with TEAMS Designer, is translated and prepared for each specific reasoner. While some of these models are basic, others are much richer and can take into account additional details and knowledge available on fault-propagation such as sequencing, timing and mode constraints, or probabilistic information. These models are generated through an automatic translation and annotation process.

The next step (B) is to design the experiment wherein our tool set allows the engineer to specify the required coverage of the test-cases in terms of the complete model or a subset of faults, including single and/or multi-fault combinations. Furthermore, the designer can specify plant operational sequences (commanded mode changes) in which these fault-scenarios need to be tested. Based on the experimental design, the fault-scenarios (C) and their associated ideal test-data (D) are auto-generated using the discretized fault-model.

Alternately, a high-fidelity simulator (F) with fault-injection capabilities is used to generate analog sensor values for each fault-scenario (generated in C). The analog data is then discretized to generate test-data. This process is further enriched by using PMA techniques to generates rich, yet small set of test-cases by perturbing fault magnitude and timing parameters (E), as well as monitoring and discretization parameters (G).

The auto-generated test-cases are then fed to each of the reasoners (H). Their outputs form the basis for the cross validation analysis (I) to get a handle on the diagnosis quality and fault-coverage taking into account the results of the sensor sensitivity analysis and test data coverage analysis. The tool suite is augmented with report generators and a number of advanced statistical analysis and visualization capabilities.

3.2. Reasoning Engines

3.2.1. TEAMS Emulator

Diagnostic reasoning with the given TEAMS model is performed using an implementation of the D-matrix diagnosis algorithm. Given a vector of discrete test results (pass, fail, unknown) and the D-matrix, four sets of failure modes are calculated, those, which are “good”, “bad”, “suspect”, or “unknown”. Failure modes in the suspect list are those, for which some tests have failed, but there has been not enough information for disambiguation.

3.2.2. TFPG

A TFPG model is a labeled directed graph where the nodes represent either failure modes, which are fault causes, or discrepancies, which are off-nominal conditions that are the effects of failure modes. Edges between nodes in the graph capture propagation of failure effects over time in the dynamic system. The model is used for fault diagnostics by collecting observations about anomalies and discrepancies (i.e., tests) in the system, and then using efficient graph search algorithms to generate fault source candidates, i.e., failure modes of components.

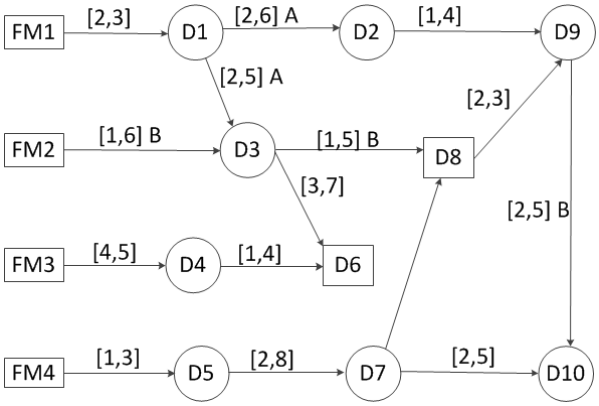

Figure 2. Example TFPG model

Figure 2 shows, as an example, a generic TFPG model. Here, rectangles represent the failure modes (FM1, FM2, ...) while circles represent OR discrepancies and squares AND discrepancies. Edges between nodes capture failure propagation in the system. The edge labels of the form: $[min, max]mode$ capture the failure propagation constraints in terms of timing interval (minimal and maximal expected times) and operational mode(s).

The TFPG modeling approach lends itself to creating system-level, hierarchical fault-propagation models of complex (physical) systems, where component failure modes are anticipated, their failure effects (discrepancies) are observable, a clear cause-effect relationship exists between failure modes and discrepancies, and the failure effects cascade across components (via material, energy, and information flows).

The TFPG reasoner (Abdelwahed, Karsai, & Biswas, 2005; Abdelwahed, Karsai, Mahadevan, & Ofsthun, 2009) employs a robust consistency based diagnosis algorithm that can account for multiple simultaneous faults while taking into account failure propagation constraints based on timing, operational mode(s), and test/effect cascading sequences. The reasoning algorithm is robust to realistic monitoring problems associated with the Tests/Alarms - false-positives, false-negatives and intermittence. The TFPG approach has been applied to and evaluated for various aerospace and industrial systems (Mahadevan & Karsai, 2000–2014; Abdelwahed et al., 2009; Hayden et al., 2006) and recently applied in the context of component-based software system (Abdelwahed, Dubey, Karsai, & Mahadevan, 2011).

3.2.3. Bayesian Networks for HM

Bayesian networks (BN) can be used for diagnosis and decision making. Domain knowledge and probabilistic information about sensor and component reliability, like MTTF, as well as failure likelihood can be easily expressed as priors. We developed a transformation of the given TEAMS model (i.e., the D-matrix) into a Bayesian network, which is inspired by (Pearl, 1988; Luo, Tu, Pattipati, Qiao, & Chigusa, 2005). Optimizations like divorcing and a subsequent translation into arithmetic circuits result in an efficient statistical reasoning engine for large models.

3.2.4. Other Reasoners

Our tool architecture allows us to incorporate additional reasoners, like, for example, HyDE (Narasimhan & Brownston, 2007), which uses simulation over simplified physical models to support diagnostic reasoning. Similarly, systems, like KATE (Goodrich, Narasimhan, Daigle, Hatfield, & Johnson, 2007), which is a generic shell for model-based simulation, monitoring and reasoning, could be added to the set of reasoners for cross validation. In these cases, however, the given TEAMS model cannot be directly translated into a model for those reasoners, as the semantic difference is too large.

3.3. Automated Scenario Generation

The Diagnostic Verification (DVER) tool for the automated scenario generation allows the user to specify the experiment design parameters relative to the appropriate discrete TFPG fault model. The user can specify the set of faults that need to be covered as part of the experiment. The coverage could include the entire model or a specific set of faults in the model. Additional parameters that can be input include: number of faults to be generated per fault scenario (e.g., single-fault, two-fault, etc.), mode change sequence to be applied, timing consideration for the fault propagation interval (e.g., minimal, random, or maximal delay), and number of missing (false-negatives), inconsistent (false-positives) or intermittent tests. Figure 3(left/center) shows a screen-shot of the DVER interface to configure the experiment.

Brute force fault scenario generation involves generating all combinations of faults from the selected list to produce single- and/or multi-fault scenarios. The n-factor algorithm used in Parametric Model Analysis (see Section 3.5) could be used to generate the minimal combinations of fault-scenarios to get the desired fault-coverage. Each generated fault-scenario includes the list of faults and their respective fault-injection times. Test-vector generation for each fault-scenario involves using the TFPG model to simulate the graph traversal starting from the fault-nodes listed in the scenario. The traversal takes into consideration any timing/mode constraint imposed by the TFPG along the fault-propagation sequence. Depending on the user selected option, it chooses the minimal/maximal or a random intermediate time (between minimal and maximal delay) for each propagation link. As the graph is traversed, the triggering time for each node is recorded. The simulator advances the clock to the next time-stamp. The nodes that are marked to be triggered at the time-stamp are then marked visited, and the traversal proceeds to mark the trig-

gering time for its child nodes. When the node corresponding to an observable discrepancy is visited, the triggering time for the test/monitor is recorded. A test once triggered is considered to be latched in that state. Any updates to the graph are applied based on the mode-changes at the specified time. The simulation/traversal process is completed when all possible discrepancy nodes are reached subject to the fault triggering time, propagation time, the mode-change sequence. The test-scenario captures the triggering time for the visited/triggered tests as well as any mode-changes. Missing tests are generated by randomly removing one or more triggered tests from the test-scenario. Inconsistent tests are generated from the set of tests that are not visited during the traversal. Intermittent tests are generated by repeatedly toggling the test-status at random times (within a specified interval).

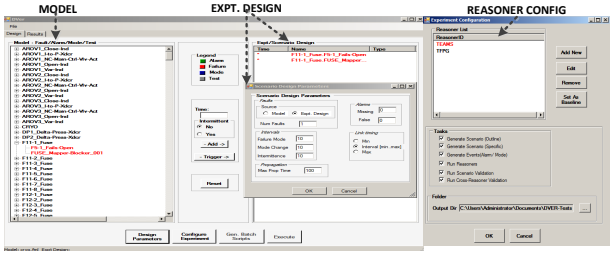

Figure 3. DVER experiment configuration

3.4. Cross Validation

Cross validation of the deployed baseline reasoner results with the results of other candidate reasoning engines facilitates analysis of the correctness, reliability, and limitations of the deployed SHM model and process. As any diagnostic model represents a simplified and abstracted model of the underlying physical plant, we leverage off an abstraction hierarchy, which simplifies the plant model towards different domains. In the current instantiation of the tool suite, the reasoners considered include TEAMS/RT (baseline deployed reasoner), TFPG, and a BN diagnoser. While the TEAMS/RT engine uses a simple dependency matrix between fault and tests in each operating mode, the TFPG and BN reasoners can take into account additional details pertaining to timing, fault propagation (sequence), and probabilistic information, respectively. In the abstraction hierarchy this would mean a step towards the time domain, and the probabilistic domain. The use of HyDE (Narasimhan & Brownston, 2007), which uses simplified physical models to support diagnostic reasoning, would correspond to yet another step in the abstraction hierarchy.

The cross-validation process starts with *Scenario Validation* - validating reasoner results against the ground truth fault-scenario to group the listed faults per hypothesis as well as across all hypotheses to identify the fault sets *Match* (true-positives – match with fault scenario) and *Extra* (false-positives – do not match with fault scenario). These are used to compute metrics that reflect the diagnosis quality in terms of *Degree of Match* (ratio of number of matched faults to total number of scenario-faults) and *Accuracy*. In cross validation, the *Match* and *Extra* sets (computed during scenario validation) of the baseline reasoner is compared against those of a candidate reasoner to compute coverage/confidence metrics that indicate the relative closeness of the correctness (match with ground truth) and accuracy (match in terms of ambiguous or erroneous results) of the two reasoners. The cross validation process is repeated against multiple reasoners to get a better assessment of the relative quality of the baseline reasoner. These results from scenario and cross validation are averaged over the desired/expected scenarios (fault subset, single/multi fault, varying fault magnitude, varying test thresholds) to get an overall assessment of the baseline diagnosis quality. Figure 3(right) shows the screen-shot of the interface for configuring reasoners.

3.5. Parametric Model Analysis

Results of system runs with parametric variations are important, among others, for robustness and sensitivity analysis. Traditionally, methods of single-parameter variation or statistical Monte Carlo techniques are used. These methods, however, fail to work on multi-failure analysis or require a large number of test cases without providing any guarantee for coverage of the parameter space. Our GUI-based PMA tool (Reed et al., 2011; Schumann, Bajwa, Berg, & Thirumalainambi, 2010; Schumann, Gundy-Burlet, Pasareanu, Menzies, & Barrett, 2009) uses an n-factor algorithm for generating perturbed fault scenarios and to modify discretization and timing parameters. For the generation of test vectors, the given perturbation range for each variable is discretized into a (small) number of bins, in our case 5, which could correspond to “almost nominal”, “lower”, “higher”, “much lower”, and “much higher”. Then the n-factor generation picks individual bins for each variable in such a way that (a) each bin of each variable is present at least once, (b) for all pairs of variables, all combinations of their pairs of bins are present. If $n = 3$, condition (b) must hold for all triples. This means that for a given n all m-ary combinations for $m \leq n$ must be present in the test set, but not necessary combinations for larger m. An n-factor algorithm makes the assumption that failures in a

Table 1. N-factor performance for different number of variables. Number of test cases and generation time (in parentheses) shown for calculations under 10 minutes.

variables	n=2	n=3	n=4	n=5
5	35(1s)	180(1s)	775(1s)	3125(1s)
10	45(1s)	309(1s)	1878(9s)	10364(480s)
15	53(1s)	390(1s)	2546(100s)	
20	58(1s)	446(1s)	3046(537s)	
50	74(1s)	629(49s)		
100	85(1s)	784(724s)		

system are only caused by $m \leq n$ triggers, and higher-order combinations of $n + 1$ or greater factors are not necessary. Experience indicates that 2 or 3 factors are usually sufficient for most applications with a substantially reduced number of test cases. Table 1 shows number of generated vectors and the generation times on a Macbook Pro.

Similar effects can be observed when using n-factor for code coverage testing. Giannakopoulou et al. (2011) report that a 3-factor set reduced the size of the test set by more than 3 orders of magnitude compared to the combinatorial exploration. Yet, only about 2% of code coverage was lost. Random test sets of the same size led to a substantially reduced coverage.

3.6. Analysis and Report Generation

Our tool suite can perform a number of analyses regarding sensor and discretization sensitivity and robustness, test data coverage, and cross validation. Since regular health models contain a large number of signals, tests, and failure modes, visualization of results is a challenge. The tool's visualization and analysis capabilities focus on three main areas: sensor sensitivity, test data coverage, and cross-validation. For our tool, we provide several levels of detail, ranging from navigable HTML documents showing the individual time series data for each sensor in a very detailed way to ROC (Receiver Operation Characteristic) curves, which summarize the overall system performance over multiple scenarios in a single plot. The user can interpret the results with a visual interface and assess the quality of the health model to the desired level of detail. We will present results of some of these analyses in the next section.

4. APPLICATION

4.1. NASA Cryo Fuel Loading

Most liquid fuel rockets use cryogenic liquid oxygen LO_2 as oxidizer, which provides high thrust per volume but is difficult to handle. Depending on the size of the rocket, extremely large large amounts of LO_2 must be pumped from a storage tank into the tank of the rocket. The different modes of operation include chill-down phases, filling (slow and fast), as well as draining the pipes, or pumping the LO_2 back into the storage tank in case the launch has been scrubbed.

Figure 4 shows a schematic overview of such a plant; the storage tank on the left-hand side contains the oxygen, from where it is pumped—using several pumps—into the rocket tank. Electrically and pneumatically operated valves control the flow through the various pipes. An operator console is used to control the loading operations and to display results of the health management system. Numerous pressure sensors, temperature sensors, and flow sensors provide real-time information about the plant status.

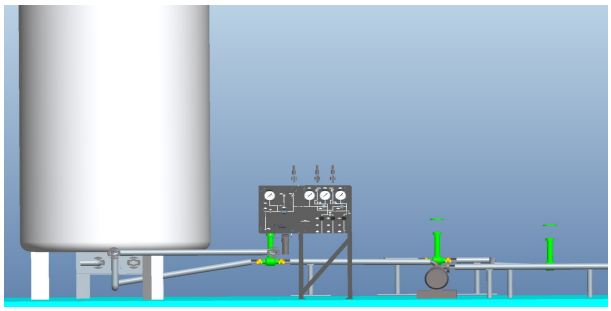

Figure 4. Generic Cryo Fuel loading plant (schematic)

4.2. Health Management and TEAMS Modeling

For this plant, a health management and diagnosis system (Goodrich et al., 2007) is being developed, using the commercial QSI TEAMS modeler and TEAMS/RT diagnosis engine. The plant is instrumented with multiple sensors for pressure, temperature, and flow. These sensor readings are captured at fixed time intervals and preprocessed in the TEAMS wrapper (Figure 5), where the signals are discretized to form test results, which are in turn used by the diagnostic engine. A single sensor can produce several test results, e.g., for a pressure sensor p, there are tests: p-nominal-in-range, p-too-high, p-too-low, etc. The outcome of each test can be "pass", "fail", or "unknown". The TEAMS model, which is based on a hierarchical multi-signal diagnosability analysis consists of several hundred tests and almost 2,000 failure modes, produced diagnosis results as sets of components (or failure modes) known to be "good", "bad", "suspect", or "unknown".

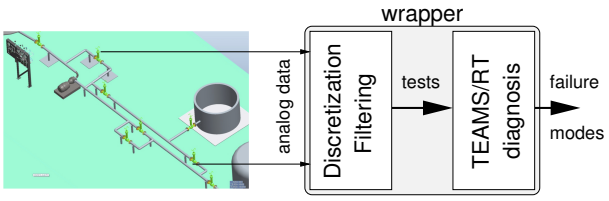

Figure 5. Cryo loading plant with TEAMS/RT wrapper

4.3. Example

For our case study, we consider a small part of a generic cryogenic fuel loading plant (Figure 6). Liquid oxygen is fed from the storage tank (left side, not shown) through the pump. The flow of LO_2 is reduced after the pump by valve V_0. Then a longer pipe transports the LO_2 to the other parts of the plant (right side of the figure). The individual pipes can be drained by means of opening V_1, V_2, or V_3. If V_4 is open, the pipes' LO_2 contents flow into a dump tank, where the liquid oxygen evaporates. This part of the plant is equipped with various pressure sensors p_1, p_2 (red) and a flow sensor f_1 (green/red).

In our operational scenario, LO_2 is pumped and V_0 is partially open to let through the fuel. A constant pressure and flow

can be measured by all sensors. At a certain time t_f, we inject a failure into the system: one of the valves V_1, V_2, or V_3 gets stuck partially open. This fault obviously causes a loss of pressure, because now a majority of the LO_2 is flowing into the dump tank.

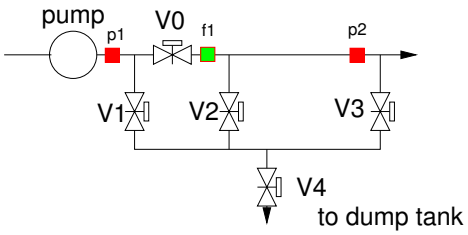

Figure 6. Schematics of small portion of the loading plant with one pump, valves V_1, V_2, V_3, V_4, pressure sensors p_1, p_2 (red), and flow sensor f_1 (green/red)

Figure 7A shows the sensor signals for the two pressure sensors p_1 (left) and p_2 (right), and the flow sensor f_1 (middle). The curves were obtained by running a physics-level Simulink simulator (see Figure 1(F)). Shown are 5 parametric variations of the failure magnitude: the valve gets stuck at $80\% \pm 20\%$. The purple lines are contrasted with a green dashed line showing the nominal (no-fault) condition. The rows of Figure 7A show the scenario, where, V_1, V_2, and V_3 fails, respectively. Graphs of the pressure and flow are shown over time. If V_1 fails, the pressure at p_1 drops almost immediately. The observed pressure drop measured at p_2 is much less and slower, because of the long pipe and the pressure reduction by V_0. The measured flow becomes considerably smaller, because LO_2 back-flows toward V_1. In contrast, when V_2 or V_3 fails, the flow actually increases, because additional LO_2 flows from the pump through the bad valves.

The comparative timing of the signals in these failure scenarios are shown in Figure 7B. The top row shows pressure development over time at location p_1, the bottom row at location p_2, respectively. The settling time ($t_{95\%}$) of the curves, belonging to each scenario can be clearly distinguished. This temporal behavior is caused by physical effects only. For a realistic plant with actual sensors, additional delay times, e.g., caused by W-LAN signal transmission, must be considered.

Our case study will focus on the analysis of these scenarios and the diagnosability of each of the failures. Specific small TEAMS models are used to discuss the tool capabilities.

4.4. Scenario Robustness and Sensitivity Analysis

Parametric Model Analysis on scenarios, shown in Figure 1(E) produces rich data sets that can be used to analyze robustness and sensitivity of the physical plant with respect to the sensors. Only if the value of a sensor changes over time in a characteristic manner when a failure occurs, its output can be potentially used for fault detection.

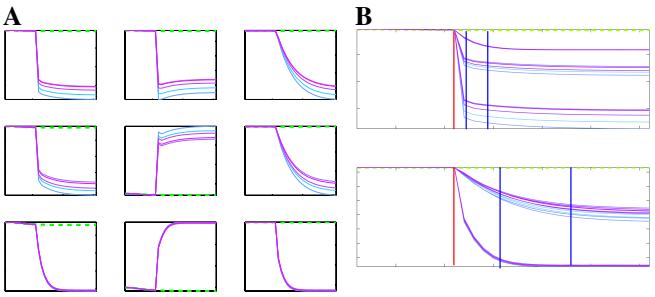

Figure 7. **A**: pressures and flow over time for scenario V_1 (top), V_2, and V_3 (bottom). Left panels show pressure at p_1, middle panels flow at f_1, and pressure at p_2 (right). **B**: delay times $t_{95\%}$ (blue) for pressures at p_1 (top) and p_2 (bottom). Fault injection at t_f shown in red. All results obtained with the Simulink plant simulator.

For a high level of detail, our tool generates navigable HTML reports, which show tables of all parametric variations of the injected faults and the time-series of all sensor outputs, contrasted to a nominal run, similar to the plots shown in Figure 7. For larger systems with many sensors a more compact representation of sensitivity results is needed. For each sensor, we therefore calculate four metrics. S_1: relative maximal deviation of the signal with respect to nominal, S_2: sensitivity of the sensor signal with respect to failure magnitude ($\partial S/\partial F$), S_3: typical shape of the curve (increase/decrease to final value, transient curve, or unspecified), and S_4: settling time $t_{95\%}$. Figure 8A shows the sensitivity for the more than 200 plant sensors for failure scenario V_1. For each sensor, its metrics are shown as star-plots. The length of each side corresponds to the normalized metrics S_1 (red), S_2 (blue), S_3 (magenta), and S_4 (cyan) — see Figure 8(center). Sensors that are not sensitive are shown as light-blue dots. Figure 8(right) displays the differences in sensitivity with respect to scenarios V_1 and V_2. Here, the number of sensitive sensors is much smaller. Sensors, which exhibit a large deviation could be used to disambiguate the failure modes relevant to these scenarios and thus could help to improve the health model.

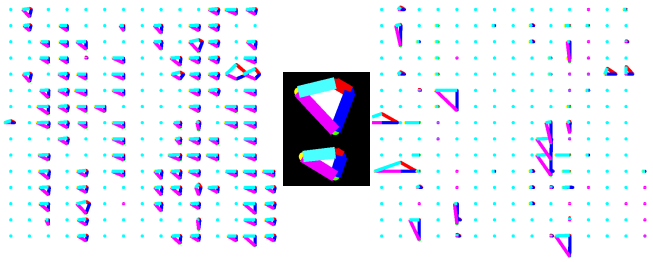

Figure 8. Sensor sensitivity for V_1 fault scenario (left). Enlarged view for 2 sensors (middle). Right panel shows the difference in sensitivity between scenario V_1 and V_2.

4.5. Threshold Robustness and sensitivity analysis

Obviously, discretization thresholds play an important role for the overall performance of the diagnosis system. Therefore, an important task is to analyze if the discretization thresholds that are provided by the domain experts are set appropriately and do not influence the diagnosis result in the presence of noise or variations in the fault magnitudes.

Figure 9 shows plots of two sensor readings S_1, S_2 over time for different failure magnitudes, as obtained from the Simulink simulator via PMA with nominal behavior (dashed green) and sensor values in different hues of blue according to the fault magnitudes. Two failures have been injected at different times t_1, t_2 (vertical purple lines). Sensor S_1 (top panel) is not very sensitive to the failure injected at t_1, but highly sensitive to failure at t_2. It is clearly visible that S_1 is sensitive with respect to the failure magnitude; different PMA runs produce different time series.

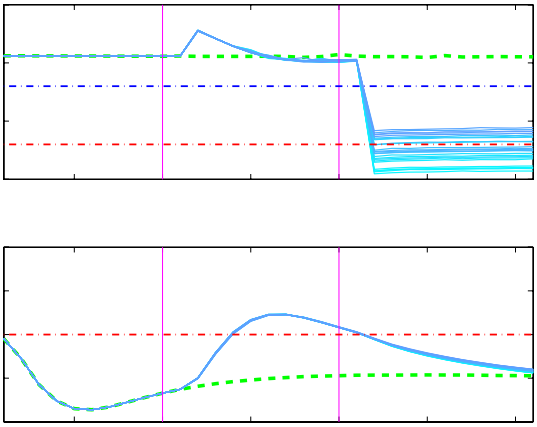

Figure 9. PMA analysis of two sensor signals S_1 (top) and S_2 (bottom)

A threshold, set to the value shown as a red dot-dashed line would result in a situation, where, depending on the actual fault magnitude (which might be subject to noise or other variations), a reliable detection might fail. On the other hand, if the threshold is set to the blue line, the off-nominal situation caused by the failure at t_2 is detected reliable regardless of the fault magnitude.

The bottom panel of Figure 9 shows the output of sensor S_2. Although it is sensitive to failure at t_1, where the nominal and off-nominal traces deviate considerably, no threshold can be found to help to detect this fault. A typical threshold (red line) would flag the fault t_1, but would also trigger during nominal operations (left part of the bottom panel). Note, that this failure causes a transient-style trace, where the value of the sensor goes back to the nominal value after some time despite the fact that this fault has been occurring. The analysis of the proper interaction between sensor signals, discretization, and reasoning results can be performed by the methods described below.

4.6. Test/Model coverage analysis

The test coverage analysis deals with understanding the quality of coverage for each test. This is done by comparing the expected test status against the realistic test status for every fault scenario. The expected test status is based on the failure-effect propagation (reachability) with the discrete fault model that is used by our reasoners (here TEAMS and TFPG). The realistic test status is obtained by thresholding the analog sensor values (from experiment or high-fidelity simulator) for the concerned fault scenarios (possibly across an interesting spectrum of fault magnitude values). The real test status generated for different thresholding criteria is compared against the expected test status to measure the test coverage quality in terms of sensitivity (true positive rate) and specificity (1-false positive rate). A higher test-coverage quality is reflected in terms of high true-positive rate and low false positive rate. The coverage quality for each thresholding criteria may be plotted and compared in an ROC (Receiver Operations Characteristics) curve. Figure 10 below shows the ROC curve obtained by changing the cut-off threshold for the test associated with pressure p_1. ROC curves are typically used to visualize

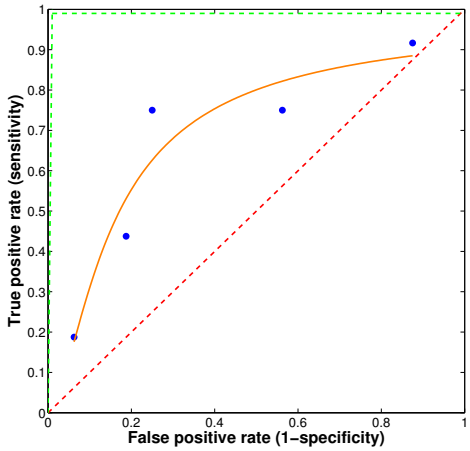

Figure 10. Coverage quality in terms of ROCs. Data points are fitted to $y = 1 - 1/((1 + (x/C)^B)^E$.

the behavior of a clustering or diagnosis algorithm. Results of experiments are shown as points of the true positive rate over the false positive rate. An ideal diagnosis system would be depicted by the green dashed line: a full true positive rate (100%) can be already reached with 0% false negatives. On the other hand, a purely random diagnosis shows up as the diagonal red line.

It is worth mentioning that the above analysis can also help capture any differences in fault-propagation (and thereby triggering of tests) between the discrete fault model (used by the reasoners) and the plant or the high-fidelity simulator. This

is especially true with tests that are never triggered in the realistic scenario but are expected by the fault-model (false negatives), as well as tests that are always triggered but not expected by the fault model (false positives). This shows up in the ROC curve as a shift in the curve along true positive rate axis and/or the false positive rate axis.

4.7. Analysis of Cross-Validation results

Analysis of the scenario validation and cross validation metrics over the specified fault scenarios helps to get a handle on the quality of diagnosis. The scenario validation process compares the faults listed in the scenario ("ground truth") against the faults reported by the diagnoser. The comparison helps identify the true positives (faults that match in the scenario and diagnosis), true negatives (scenario-faults that are not reported by the diagnoser), and false positives (faults listed by the diagnoser that are not part of the scenario). The quality of the results is expressed in terms of *Match* (percentage of true positives among the faults listed in scenario) and *Accuracy* (percentage of true positives among all faults listed by the reasoner). While *Match* is a measure of the ability of the reasoner to identify the real fault sources, *Accuracy* is a measure of the ambiguities listed by the reasoner.

Table 2 captures the results of scenario validation for our case study. It shows *Match* and *Accuracy* of three reasoners—TEAMS emulator, TFPG using the same fault-propagation model as TEAMS, and TFPG*, a TFPG reasoner using an updated fault-propagation model, which includes fault propagation times and fault propagation sequences based on the results of failure analysis shown in Figure 7. Specifically, the TFPG* model has been updated with (a) fault propagation time and (b) a propagation link between p_2 and p_1 for fault from V_3. Table 2 shows the results for ideal test vectors the TEAMS and TFPG model exhibit similar performance, but the TFPG with the updated model has a far greater accuracy (fewer ambiguities). In case of the realistic test vectors that include missing alarms, false alarms, and intermittents, the TFPG reasoner has a slightly higher accuracy probably related to the way intermittents are handled. The TFPG reasoner identifies intermittence and waits for the tests to stabilize before updating results. The TEAMS emulator, on the other hand, starts afresh with every time-stamp. This could also explain the slight decrease in *Match* (compared to ideal) for the TEAMS emulator, as it does not report any faults when all alarms disappear while exhibiting intermittence.

In computing the cross validation metrics, the candidate reasoner results for ground truth and the baseline reasoner results are compared to identity, for each result, the true positive (faults listed by both reasoners), true negative (faults listed by candidate and not by baseline), and false positive (faults listed by baseline and not by candidate). These help analyze the degree of *Match* between the reasoners in identifying faults (*Match Scenario*) and in eliminating ambiguities (*Match Extra*). These metrics help establish the accuracy of the baseline deployed reasoner relative to the candidate reasoners.

Table 3 captures these metrics for the baseline TEAMS emulator relative to the two candidate TFPG reasoners. The high numbers for the *Match Scenario* reflect the closeness between the baseline and candidate reasoner in identifying the source of the fault. A lower *Match Extra* in case of TFPG with the update model reveals that the candidate reasoner has a tighter ambiguity set than the baseline reasoner.

Table 2. Scenario Validation

Reasoner	Ideal Test-Vectors		Realistic Test-Vector	
	Match	Accuracy	Match	Accuracy
TEAMS Emulator	1	0.66	0.9	0.47
TFPG	1	0.66	1	0.59
TFPG*	1	1	1	0.83

Table 3. Cross Validation (baseline - TEAMS Emulator)

Reasoner	Ideal Test-Vectors		Realistic Test-Vector	
	Match Scenario	Match Extra	Match Scenario	Match Extra
TFPG	1	1	0.93	0.79
TFPG*	1	0.33	0.93	0.45

Analysis with the scenario validation metric is important to understand the relative performance of different reasoners. A consistently poor scenario validation metric across all reasoners could indicate a problem with the fault model (inability to isolate a fault), sensor placement, or tolerance boundaries on test coverage. Alternatively, the metric could indicate the effectiveness of one reasoner in certain cases (fault scenarios/modes/robustness to test coverage changes). On the other hand, cross validation helps benchmark the performance of the baseline relative to each candidate reasoner. This would be useful when the real source of the fault is not known and the candidate reasoners have to be used to predict the performance of the baseline reasoner. Since each candidate reasoner might have their own limitations, it is better to cross-validate against a bank of candidate reasoners. Furthermore, analysis over different subsets of faults helps identify where the baseline reasoner might be lacking when compared to the candidate reasoner. This analysis makes it possible to improve the performance of the baseline by adding suitable tests or pseudo-tests.

5. CONCLUSIONS AND FUTURE WORK

For V&V it is essential to ensure robustness and reliability of a health management systems, even more if it is to be deployed in a safety and mission critical environment. In this paper, we have presented a tool set to support V&V of

TEAMS health management systems by employing the paradigms of *cross validation*, where diagnosis results of the TEAMS model are compared with results of other, more advanced reasoners, *automatic fault scenario generation* to support extensive testing and coverage analysis, and *parametric model analysis* to enrich test sets for robustness and sensitivity analysis. We used, as an example, a subsystem of a large NASA cryogenic fuel loading system to demonstrate tool capabilities and to present initial results. A number of specific coverage metrics have been introduced for assessing model quality and model coverage during pre-deployment V&V.

In this paper we have described scaling properties of core algorithms of our integrated V&V tools. For example, n-factor combinatorial test generation scales well with increasing dimension. To provide users with succinct yet meaningful metrics to assess validation, we provide a summary analysis in terms of scenario validation (match, accuracy), cross-validation (match scenario, match extra), and ROC curves. We expect to present our results of using this tool suite on a large system with a rich set of failure effect propagation. In future work we will extend our V&V tool suite to include advanced machine learning algorithms and further statistical analysis in order to provide deeper analysis and to improve quality and robustness of scenario generation and parameter perturbation.

ACKNOWLEDGMENTS

This work was in part supported by NASA's Software Assurance Research Program (SARP), project "Advanced Tools and Techniques for V&V of IVHM Systems". Peter Berg provided a Matlab implementation for TEAMS reasoning.

REFERENCES

Abdelwahed, S., Dubey, A., Karsai, G., & Mahadevan, N. (2011). Model-based tools and techniques for real-time system and software health management. *Machine Learning and Knowledge Discovery for Engineering Systems Health Management*, 285.

Abdelwahed, S., Karsai, G., & Biswas, G. (2005). A consistency-based robust diagnosis approach for temporal causal systems. In *16th International Workshop on Principles of Diagnosis* (pp. 73–79).

Abdelwahed, S., Karsai, G., Mahadevan, N., & Ofsthun, S. C. (2009). Practical considerations in systems diagnosis using timed failure propagation graph models. *Instrumentation and Measurement, IEEE Transactions on*, *58*(2), 240–247.

Giannakopoulou, D., Bushnell, D., Schumann, J., Erzberger, H., & Here, K. (2011). Formal testing for separation assurance. *Ann. Math. Artif. Intell.*, *63*(1), 5–30.

Goodrich, C., Narasimhan, S., Daigle, M., Hatfield, W., & Johnson, R. (2007). *Applying model-based diagnosis to a rapid propellant loading system.*

Gundy-Burlet, K., Schumann, J., Menzies, T., & Barrett, T. (2008). Parametric Analysis of ANTARES Re-entry Guidance Algorithms using advanced Test Generation and Data Analysis. In *Proc. i-SAIRAS 2008.*

Hayden, S., Oza, N., Mah, R., Mackey, R., Narasimhan, S., Karsai, G., Shirley, M. (2006). *Diagnostic technology evaluation report for on-board crew launch vehicle* (Tech. Rep.). NASA.

Lindsey, A. E., & Pecheur, C. (2004). Simulation-based verification of autonomous controllers via Livingstone Pathfinder. In K. Jensen & A. Podelski (Eds.), *Proceedings TACAS 2004* (Vol. 2988, pp. 357–371). Springer.

Luo, J., Tu, H., Pattipati, K., Qiao, L., & Chigusa, S. (2005). Graphical models for diagnosis knowledge representation and inference. In *Autotestcon. IEEE* (p. 483-489).

Mahadevan, N., & Karsai, G. (2000–2014). *Fact tool suite.* `https://fact.isis.vanderbilt.edu/`.

Narasimhan, S., & Brownston, L. (2007). HyDE – a general framework for stochastic and hybrid model-based diagnosis. In *In Proc. of 18th international workshop on principles of diagnosis (DX '07)* (pp. 162–169).

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of plausible inference* Morgan Kaufmann: .

Reed, E., Schumann, J., & Mengshoel, O. (2011). Verification and validation of system health management models using parametric testing. *Proc. of Infotech@ Aerospace 2011*.

RTCA. (2011). *Do-178c: Software considerations in airborne systems and equipment certification.* Retrieved from `http://www.rtca.org`

Schumann, J., Bajwa, A., Berg, P., & Thirumalainambi, R. (2010). Parametric testing of launch vehicle FDDR models. In *AIAA Space.*

Schumann, J., Gundy-Burlet, K., Pasareanu, C., Menzies, T., & Barrett, T. (2009). Software V&V support by parametric analysis of large software simulation systems. In *Proc. IEEE Aerospace.* IEEE Press.

Schumann, J., Srivastava, A., & Mengshoel, O. (2010). Who guards the guardians? — toward V&V of health management software. In *RV 2010.* Springer.

Schwabacher, M. A., Feather, M. S., & Markosian, L. Z. (2008). Verification and validation of advanced fault detection, isolation and recovery for a NASA space system. In *Proc. PHM 2008.*

Srivastava, A., & Schumann, J. (2013). Software health management: A necessity for safety critical systems. *Innovations in Systems and SW Eng.*, *9*(4), 219–233.